UNITED STATES PATENT APPLICATION FOR

CIRCUIT-LEVEL MEMORY AND COMBINATIONAL BLOCK MODELING

Inventor:
Adrian J. Isles

CERTIFICATE OF MAILING BY "EXPRESS MAIL" UNDER 37 C.F.R. § 1.10

"Express Mail" mailing label number: EL 504218404 US
Date of Mailing: June 2, 2000

I hereby certify that this correspondence is being deposited with the United States Postal Service, utilizing the "Express Mail Post Office to Addressee" service addressed to **Box PATENT APPLICATION, Commissioner for Patents, Washington, D.C. 20231** and mailed on the above Date of Mailing with the above "Express Mail" mailing label number.

Johann S. Mercado
Signature Date: June 2, 2000

CIRCUIT-LEVEL MEMORY AND COMBINATIONAL BLOCK MODELING

Inventor: Adrian J. Isles

BACKGROUND OF THE INVENTION

Field of the Invention

The present invention relates to electronic design automation (EDA) systems. More particularly, the present invention relates to memory modeling for use with EDA tools.

Description of the Related Art

The design process for integrated circuits typically involves multiple transformations of a design from an initial idea to a functional, manufacturable product. A chip architect or designer begins with a design idea and then generates a corresponding behavioral definition of the design. The behavioral design results in a flow chart or a flow graph using which the designer can design the system data path and the registers and logic units necessary for implementation of the design. After the designer designs buses for coordinating and controlling the movement of data between registers and logic units, the data registers, buses, logic units, and their controlling hardware are implemented using logic gates and flip-flops. The result of this design stage is a netlist of gates and flip-flops. The netlist can be used to create a simulation model of the design to verify the design before

provide the information needed by a routing software package to complete the actual design. The netlist of gates and flip-flops is thus transformed into a transistor list or layout and gates and flip-flops are replaced with their transistor equivalents or library cells. Timing and loading issues are also addressed during this cell and transistor selection process. Finally, the manufacturing process begins when the transistor list is implemented in a programmable logic device such as an FPGA or when the layout specification is used to generate masks for integrated circuit fabrication.

EDA tools improve upon this design process by permitting electronic circuit designers to more quickly and inexpensively design and verify their designs. Figure 1 illustrates a typical design approach using EDA tools. The designer initially supplies a logic synthesis tool 120 with a high level language description 110 of the design and the logic synthesis tool 120 reduces the high level language description 110 to a low level or gate level description 130 of the design. Finally, verification or simulation of the design is performed by an engine 140 using a set of properties 150 or behaviors as an input to determine whether, and to what extent, the design described by HDL description 110 satisfies the properties 150. The properties 150 are based on a functional specification for the design being verified or simulated. A more detailed discussion of design verification and the use of properties can be found in U.S. Patent Application Serial No. 09/447,085, filed November 22, 1999, entitled "Static Coverage Analysis," incorporated herein by reference.

The description of the design idea is typically written in a high-level hardware description language ("HDLs") such as VHDL or Verilog. HDLs provide formats for representing the output of the various design stages described above and are thus used to create circuits at various levels of abstraction including gate-level descriptions of functional blocks and high-level descriptions of complete systems. HDLs provide a convenient format for the representation of functional and wiring details of designs and may represent various hardware components at one or more levels of abstraction. HDLs can be used to model many different kinds of hardware components or electronic circuits. VHDL and Verilog are commonly used to model circuits ranging from ALUs, arithmetic blocks, bus arbiters, bus interfaces, cache controllers, data paths, dual-phase clocks, instruction and address decoders, pipelines, reset circuits, sequencers, and state machines.

The design approach of Figure 1 has also been previously used to model, synthesize, and verify memory circuit designs. However, prior art memory models suffered from two major disadvantages. First, they modeled every location of the memory even if only a subset of locations of the memory were actually used by the design. For example, if the designer is modeling a RAM having, for example, 1024 locations, the designer must provide a functional description for each location of the memory even if the design only accessed a small portion of the memory locations. The resulting memory model was inefficient and wasted valuable design resources. Second, memory models from one EDA tools vendor can typically only be

used with simulation or verification engines from the same vendor. In other words, the choice to use a particular prior art memory model necessitated a particular verification or simulation engine. This lack of interoperability greatly limits the utility of prior art memory models.

Accordingly, there is a need for a means for modeling physical memory that more efficiently describes only those portions of a physical memory that are used by a given design. There is a further need for a memory model that is independent of the underlying simulation or verification engine and is thus interoperable with various simulation or verification engines.

SUMMARY OF THE INVENTION

The present invention, roughly described, provides a method for modeling a physical memory for use in an electronic circuit design where memory write operations to the physical memory and memory read operations from the physical memory are modeled in a lookup table. The size or the number of entries in the lookup table is limited by a total number of memory operations that can occur over a given number of clock cycles. In one embodiment, the upper bound is equal to the total number of memory operations that can be performed per clock cycle times the number of clock cycles plus any number of memory read operations specified in the properties. In another embodiment, the contents of the lookup table can, at any time, be initialized to a constant value or to an arbitrary initial value.

The present invention can be implemented using software, hardware, or a combination of software and hardware. When all or portions of the present invention are implemented in software, that software can reside on a processor readable storage medium. Examples of an appropriate processor readable storage medium include a floppy disk, hard disk, CD-ROM, memory IC, etc. The hardware used to implement the present invention includes an output device (e.g. a monitor or printer), an input device (e.g. a keyboard, pointing device, etc.), a processor in communication with the output device and processor readable storage medium in communication with the processor.

The processor readable storage medium stores code capable of programming the processor to perform the steps to implement the present invention. In one embodiment, the present invention may comprise a dedicated processor including processor instructions for performing the steps the implement the present invention. In another embodiment, the present invention can be implemented on a web page on the Internet or on a server that can be accessed over communication lines. These and other objects and advantages of the invention will appear more clearly from the following detailed description in which the preferred embodiment of the invention has been set forth in conjunction with the drawings.

<u>BRIEF DESCRIPTION OF THE DRAWINGS</u>

Figure 1 is a block diagram overview of a prior art EDA tool.

Figure 2 is a flowchart of a memory write operation in accordance with the present invention.

Figure 3 is a flowchart of a memory read operation in accordance with the present invention.

Figure 4 is an exemplar memory write operation according to the memory model of the present invention.

Figure 5 is an exemplar memory read operation according to the memory model of the present invention.

Figure 6 is a block diagram illustrating the memory model of the present invention as used in an existing EDA tool.

Figure 6A illustrates another embodiment of the present invention.

Figure 6B is a flowchart illustrating another embodiment of the present invention.

Figure 7 is a high level block diagram of an exemplar general purpose computer system which can be used to implement the present invention.

DETAILED DESCRIPTION

Figure 2 illustrates a memory model that can be implemented as lookup table 250, for use in modeling a physical memory 210 of an electronic circuit design. While physical memory 210 contains "i" locations, data has been written to only four of those locations. In physical memory 210, for example, the electronic circuit design has written a plurality of write data bits represented by D_2 to write address A_2. A plurality of write data bits

represented by D_4 have been written to write address A_4. A plurality of write data bits represented by D_6 have been written to write address A_6. A plurality of write data bits represented by D_i have been written to write address A_i. Rather than modeling all "i" locations of physical memory 210, the present invention models only those locations of physical memory 210 that have been accessed by the electronic circuit design. Lookup table 250 is created to perform this task. Lookup table 250 comprises an address field 255, a data field 260, and a valid bit field 265. While not drawn, lookup table 250 may also include a write enable bit that can be asserted to signify a memory write operation. It should also be noted that all valid bits of lookup table 250 in valid bit field 265 are initially in an unasserted logic state.

The total number of entries in lookup table 250 ("n") is defined by the total number of memory operations that can occur over a given number of clock cycles. More specifically, the total number of entries in lookup table 250 is greater than or equal to the total number of memory operations that can occur in physical memory 210 over a given number of clock cycles (or over the period of time in which physical memory 210 is modeled or over the period in which the electronic circuit comprising physical memory 210 is simulated or verified).

This upper bound is computed by determining the number of read and write ports of the physical memory that is to be modeled, determining a total number of memory operations that can be performed per clock cycle, and

multiplying the total number of memory operations per clock cycle with the number of clock cycles. For example, if the physical memory that is to be modeled contains two read ports and two write ports, then a maximum of four memory operations can occur in each clock cycle (i.e., two times two). Thus, if the period of time over which the physical memory is to be modeled is, for example, five cycles, then the total number of memory operations that can occur over this period of time is twenty (i.e., four memory operations per clock cycle times five cycles).

It should be noted that this upper bound can also be affected by additional memory read operations that may be specified in one or more properties 150 of Figure 1. In such a case, the number of additional memory read operations would have to be added to the total number of memory operations previously calculated. For example, if a given property specified, for example, two additional memory read operations, then the total number of memory read that can occur over the period of time in which the physical memory is modeled is twenty-two (i.e., twenty plus two).

The first steps in modeling a memory write operation, WRITE(WADDR, WDATA), to physical memory 210 includes the steps of receiving write addresses (WADDR) of physical memory 210 to which data (WDATA) has been written, receiving WDATA written to physical memory 210 at the respective write addressees, and asserting a write enable signal. The next step is to check each entry in lookup table 250 to find an entry where the valid bit is asserted and the corresponding address in address field 255 in

lookup table 250 of the entry is equivalent to the WADDR. If such an entry exists, WDATA is written to data field 260 of that entry. However, if such an entry in lookup table 250 does not exist, the present invention finds an entry in lookup table 250 with an unasserted valid bit and places WADDR in address field 255 of that entry and WDATA is placed in data field 260 of that entry.

Consider, for example, memory write operation 215 to physical memory 210, WRITE(A_2, D_2), where A_2 represents a write address and D_2 represents data being written to that write address. Given the write address (A_2), the write data (D_2), and the assertion of a write enable bit, the first step in modeling this memory write operation to physical memory 210 is to check each entry in lookup table 250 to find an entry where the valid bit is asserted and the address in the address field of that entry matches the write address (A_2). In this case, D_2 is written to data field 260 of entry number 1 of lookup table 250 since the valid bit of entry number 1 is asserted and A_2 is contained in address field 255 of entry number 1. Memory write operations 220 and 225 to physical memory 210 can be similarly modeled. That is, since the valid bit of entry number 2 is asserted and write address A_4 is found in address field 255 of entry number 2, the previous contents in data field 260 of entry number 2 are overwritten with write data D_4. Also, since the valid bit of entry number 3 of lookup table 250 is asserted and write address A_6 is found in address field 255 of entry number 3, write data D_6 is

written to the data field 260 of entry number 3 (therein overwriting any data that may have previously been in data field 260 of entry number 3).

Consider now memory operation 230 in which write data D_i is written to write address A_i of physical memory 210. If after searching through all the entries in lookup table 250 and there are no entries where the valid bit is asserted and the corresponding address in address field 255 is equal to A_i, then the present invention, for example, finds an entry "n", where the valid bit is unasserted and places the write address A_i in address field 255 of entry "n", the write data D_i in data field 260 of entry "n," and asserts the valid bit of entry "n". All memory write operations to physical memory 210 can thus be modeled in lookup table 250.

Figure 3 illustrates an exemplar memory read operation in accordance with the memory model of the present invention. The steps for modeling a memory read operation to physical memory 210 are similar to those of the memory write operation. Given a particular read address, RADDR, the present invention searches lookup table 250 for entries where the valid bit is asserted and address field 255 for that entry contains the RADDR. If such an entry exists in lookup table 250, the present invention returns the data in data field 260 corresponding to that entry. However, if such an entry is not found in lookup table 250, the present invention first finds an entry in lookup table 250 where the valid bit is unasserted. Next, the read address RADDR is placed in address field 255 of that entry and an arbitrary data value is assigned to data field 260 of that entry. The valid bit of that entry is then

asserted and the arbitrary date value assigned to data field 260 is then returned. In one embodiment of the present invention, an initial value to which the contents of lookup table 250 have been previously initialized may be returned instead of the arbitrary data value.

Consider, for example, memory read operation 310, READ(A_2), where the contents of the memory location in physical memory 210 identified by read address A_2 are now being read. The present invention models memory read operation 310 by first searching each entry in lookup table 250 for an entry where the valid bit is asserted and address field 255 of that entry contains read address A_2. In this example, the valid bit of entry 1 is asserted and address field 255 of entry 1 contains read address A_2. The contents of data field 260 of entry number 1, D_2, are thus returned. The present invention similarly models memory read operation 320, READ(A_6), by first searching lookup table 250 for an entry where the valid bit is asserted and address field 255 of that entry contains read address A_5. Entry number 3 of lookup table 250 satisfies these conditions and the contents of data field 260 of entry number 3, D_6, are subsequently returned.

Now consider memory read operation 315, READ(A_3), where data corresponding to read address A_3 is being read from physical memory 210. In this scenario, there are no entries in lookup table 250 where the valid bit is asserted and address field 255 contains read address A_3. Consequently, the present invention models memory read operation 315 by first finding an entry in lookup table 250, entry "n-1," where the valid bit is unasserted and

assigns read address A_3 to address field 255 of entry number "n-1," and assigns an arbitrary data value to data field 260 of entry number "n-1." The present invention then asserts the valid bit for entry number "n-1" and subsequently returns the arbitrary data value. As before, in one embodiment of the present invention, an initial value to which the contents of lookup table 250 may have previously initialized may be returned instead of the arbitrary data value. All memory read operations from physical memory 210 can thus also be modeled by lookup table 250.

Figure 4 is a flowchart of a memory write operation in accordance with the memory model of the present invention. In step 410, the memory write address, WADDR, and the write data, WDATA, corresponding to a memory write operation to physical memory 210 is first received. In step 415, lookup table 250 of Figures 2 and 3 is searched for entries where the valid bit is asserted and the corresponding address in the address field of that entry is equal to the write address WADDR. If such an entry in lookup table 250 exists, write data WDATA is written to data field 260 corresponding to that entry in step 420. However, if there are no entries in lookup table 250 where the valid bit is asserted and the corresponding address in address field 255 is equal to write address WADDR, then, in step 425, the present invention finds an entry in lookup table 250 with an unasserted valid bit. In step 430, write address WADDR is placed in address field 255 of that entry. In step 435, write data WDATA is placed in data field 260 of that entry and the valid bit of that entry is asserted in step 440. These steps are performed

for each memory write operation to physical memory 210 and the resulting lookup table models all write operations to physical memory 210.

Figure 5 is a flowchart of a memory read operation in accordance with the memory model of the present invention. Modeling a memory read operation from physical memory 210 begins in step 510 with read address RADDR being received. In step 515, the present invention searches lookup table 250 for entries where the valid bit is asserted and address field 255 contains the read address RADDR. If such an entry in lookup table 250 exists, the data in data field 260 of that entry is returned in step 520. However, if lookup table 250 does not contain an entry where the valid bit is asserted and address field 255 does not contain read address RADDR, then, in step 525, the present invention finds an entry in lookup table 250 with an unasserted valid bit. In step 530, read address RADDR is placed in address field 255 of that entry and, in step 535, an arbitrary data value is assigned to data field 260 of that entry. In step 540, the valid bit for that entry is asserted and, in step 545, the arbitrary data value in data field 260 of that entry is returned. These steps are performed for each memory read operation from physical memory 210 and the resulting lookup table 250 models all read operations from physical memory 210.

Figure 6 illustrates how the memory model of the present invention is used in an EDA tool. For a given electronic circuit design having a physical memory, the process of simulating/verifying the design begins with the creation of a circuit description 620 of the design. While this circuit

description could be an HDL description, any circuit description of the design will suffice. Circuit description 620 would then be inputted to logic synthesis tool 630. Logic synthesis tool 630 reduces circuit description 620 to a gate level description 640 of the electronic circuit design. That portion of gate level description 640 relating to the physical memory is then replaced with lookup table 645 wherein lookup table 645 effectively models all read and write ports of the physical memory. That is, all memory write operations to the physical memory and all memory read operations from the physical memory are modeled in lookup table 645 in accordance with the steps outlined in Figures 4 and 5, respectively. Finally, verification or simulation of the electronic circuit design is performed by a verification or a simulation engine 650 together with a set of properties 660 or behaviors as an input to determine whether, and to what extent, the electronic circuit design satisfies properties 660.

Thus, in accordance with the present invention, all read and write operations to and from any physical memory can be modeled by a lookup table. Aside from being a more efficient way to model only those locations of the physical memory that are accessed by a given design, there are two other significant advantages in using the lookup table approach of the present invention. First, the lookup table approach enables a physical memory to be modeled independent of the underlying simulation or verification engine. This is significant in that the lookup table approach of the present invention allows circuit designers and chip architects to create

memory models that can be used with ALL existing simulation or verification engines.

Second, and perhaps more importantly, Figure 6A illustrates another embodiment of the present invention. The lookup table approach of the present invention can be used to model any uninterpreted combinational block 666 or any combinational function, having "m" inputs and "n" outputs, of a given design comprising a circuit element 664 and a circuit element 668. In other words, the lookup table approach of the present invention can be used to model so called "black boxes" or uninterpreted portions of a design. An uninterpreted combinational block 666 or a combinational function can be modeled using a lookup table in much the same way that a memory read operation from physical memory 210 is modeled in lookup table 250 (see Figures 3 and 5). The only difference is that the read address is now the argument(s) of the combinational functions.

Figure 6B is a flowchart illustrating a method of modeling an uninterpreted combinational block or function using a lookup table. In step 670, the lookup table is initialized. Step 670 may include the steps of initializing the contents of the lookup table to some initial value, and setting all valid bits of the lookup table to some initial state (e.g., unasserted). In step 672, the present invention searches the lookup table for entries where the valid bit is asserted and an "address" field contains the argument(s) of the combinational block or function.

If such an entry in the lookup table exists, then, in step 676, the combinational block or function is modeled using a first set of process step(s). The first set of process step(s) may include a step where a data value corresponding to the argument is returned if an entry in the lookup table comprises the argument and a valid bit of that entry is asserted. However, if such an entry does not exist, then, in step 674, the combinational block or function is modeled using a second set of process step(s). The second set of process step(s) may include the steps where an entry in the lookup table with an unasserted valid bit is found, the argument is placed in the "address" field of that entry, an arbitrary data value is assigned to that entry corresponding to the argument, the valid bit of that entry is asserted, and the arbitrary data value is returned. In step 680, the resulting model of the uninterpreted combinational block can be provided as an input to a gate level description of the design. The design containing the uninterpreted combinational block can then be simulated or verified using an engine (not shown).

For example, a combinational function, *f*(X), representing a given uninterpreted combinational block of a design where "X" is the argument of the function, can be modeled in a lookup table using a READ(X) operation. As with the memory write operation described in Figures 3 and 5, for a given clock cycle, a READ(X) operation will return a data value corresponding to argument "X" if an entry in the lookup table comprises argument "X" in an "address" field and a valid bit of that entry is asserted. If, however, no

entries in the lookup table comprise argument "X" and an asserted valid bit, then argument "X" is placed in an entry of the lookup table with an unasserted valid bit, an arbitrary data value corresponding to argument "X" is returned, and the valid bit of that entry is asserted.

This process is repeated for as many clock cycles as are required to verify or simulate the design containing the uninterpreted combinational block. While this exemplar function contains only one argument "X," the lookup table approach of the present invention can be used to model combinational blocks or functions having multiple arguments. The resulting model of any uninterpreted combinational blocks or black boxes of a design will also be engine independent. That is, the resulting model can be used with all existing simulation or verification engines.

Figure 7 illustrates a high-level block diagram of a general purpose computer system which can be used to implement the present invention. The computer system 710 of Figure 7 includes a processor unit 712 and main memory 714. Processor unit 712 may contain a single microprocessor, or may contain a plurality of microprocessors for configuring the computer system as a multi-processor system. Main memory 714 stores, in part, instructions and data for execution by processor unit 712. If the present invention is wholly or partially implemented in software, main memory 714 stores the executable code when in operation. Main memory 714 may include banks of dynamic random access memory (DRAM) as well as high-speed cache memories.

The computer system 710 of Figure 7 further includes a mass storage device 716, peripheral device(s) 718, input device(s) 720, portable storage medium drive(s) 722, a graphics subsystem 724, an output display 726, and output devices 732. For purposes of simplicity, the components in computer system 710 are shown in Figure 7 as being connected via a single bus 728. However, computer system 710 may be connected through one or more data transport means. For example, processor unit 712 and main memory 714 may be connected via a local microprocessor bus, and the mass storage device 716, peripheral device(s) 718, portable storage medium drive(s) 722, graphics subsystem 724, and output devices 732 may be connected via one or more input/output (I/O) buses. Mass storage device 716, which may be implemented with a magnetic disk drive or an optical disk drive, is a non-volatile storage device for storing data and instructions for use by processor unit 712. In one embodiment, mass storage device 16 stores the system software for determining a path for purposes of loading to main memory 714.

Portable storage medium drive 722 operates in conjunction with a portable non-volatile storage medium, such as a floppy disk, to input and output data and code to and from computer system 710. In one embodiment, the system software for determining a path is stored on such a portable medium, and is input to the computer system 710 via the portable storage medium drive 722. Peripheral device(s) 718 may include any type of computer support device, such as an input/output (I/O) interface, to add

additional functionality to the computer system 710. For example, peripheral device(s) 718 may include a network interface card for interfacing computer system 710 to a network, a modem, etc.

Input device(s) 720 provide a portion of the user interface for a user of computer system 710. Input device(s) 720 may include an alpha-numeric keypad for inputting alpha-numeric and other key information, or a cursor control device, such as a mouse, a trackball, stylus, or cursor direction keys. In order to display textual and graphical information, computer system 710 contains graphics subsystem 724 and the output display 726. Output display 726 may include a cathode ray tube (CRT) display, liquid crystal display (LCD) or other suitable display device. Graphics subsystem 724 receives textual and graphical information, and processes the information for output to output display 726.

Output display 726 can be used to report the results of a sign text computation. Output devices 732 provide another means for reporting the results of a sign text computation. Output devices 732 may include a printer, a personal digital assistant (PDA), a modem, a cellular telephone capable of transmitting and receiving text messages, audio speakers, or any other device to which the results of the sign text The components contained in computer system 710 are those typically found in general purpose computer systems, and are intended to represent a broad category of such computer components that are well known in the art.

The components contained in the computer system of Figure 7 are those typically found in general purpose computer systems, and are intended to represent a broad category of such computer components that are well known in the art. Thus, the computer system of Figure 7 can be a personal computer, workstation, minicomputer, mainframe computer, etc. The computer can also include different bus configurations, networked platforms, multi-processor platforms, etc. Various operating systems can be used including Unix, Linux, Windows, Macintosh OS, and other suitable operating systems.

The foregoing description of preferred embodiments of the present invention has been provided for the purposes of illustration and description. It is not intended to be exhaustive or to limit the invention to the precise forms disclosed. Many modifications and variations will be apparent to practitioners skilled in this art. The described embodiments were chosen and described in order to best explain the principles of the invention and its practical application to thereby enable others skilled in the art to understand the invention for various embodiments and with various modifications as are suited to the particular use contemplated. It is intended that the scope of the invention be defined by the following claims and their equivalents.